**JavaBasic: Real Estate Listing Management**

class RealEstateListing implements IRealEstateListing {

private int ID;

private String title;

private String description;

private int price;

private String location;

@Override

public void setID(int ID) {

this.ID = ID;

}

@Override

public int getID() {

return ID;

}

@Override

public void setTitle(String title) {

this.title = title;

}

@Override

public String getTitle() {

return title;

}

@Override

public void setDescription(String description) {

this.description = description;

}

@Override

public String getDescription() {

return description;

}

@Override

public void setPrice(int price) {

this.price = price;

}

@Override

public int getPrice() {

return price;

}

@Override

public void setLocation(String location) {

this.location = location;

}

@Override

public String getLocation() {

return location;

}

}

class RealEstateApp implements IRealEstateApp {

private List<IRealEstateListing> listings = new ArrayList<>();

@Override

public void addListing(IRealEstateListing listing) {

listings.add(listing);

}

@Override

public void removeListing(int listingID) {

Iterator<IRealEstateListing> iterator = listings.iterator();

while (iterator.hasNext()) {

IRealEstateListing listing = iterator.next();

if (listing.getID() == listingID) {

iterator.remove();

break;

}

}

}

@Override

public void updateListing(IRealEstateListing listing) {

for (IRealEstateListing existingListing : listings) {

if (existingListing.getID() == listing.getID()) {

existingListing.setTitle(listing.getTitle());

existingListing.setDescription(listing.getDescription());

existingListing.setPrice(listing.getPrice());

existingListing.setLocation(listing.getLocation());

break;

}

}

}

@Override

public List<IRealEstateListing> getListings() {

return listings;

}

@Override

public List<IRealEstateListing> getListingsByLocation(String location) {

List<IRealEstateListing> filteredListings = new ArrayList<>();

for (IRealEstateListing listing : listings) {

if (listing.getLocation().equals(location)) {

filteredListings.add(listing);

}

}

return filteredListings;

}

@Override

public List<IRealEstateListing> getListingsByPriceRange(int minPrice, int maxPrice) {

List<IRealEstateListing> filteredListings = new ArrayList<>();

for (IRealEstateListing listing : listings) {

int price = listing.getPrice();

if (price >= minPrice && price <= maxPrice) {

filteredListings.add(listing);

}

}

return filteredListings;

}

}

**Problem Solving**

**Academic Decathlon**

import java.util.Arrays;

public class Main {

public static int findMaxTeamSize(int[] skills) {

Arrays.sort(skills); // Sort the skills array

int n = skills.length;

int i = 0;

int ans = 0;

while (i < n) {

int j = i + 1;

while (j < n && skills[j] - skills[j - 1] <= 1) {

j++;

}

ans = Math.max(ans, j - i);

i = j;

}

return ans;

}

public static void main(String[] args) {

int[] skills = {1, 3, 5, 2, 4};

System.out.println("Max team size: " + findMaxTeamSize(skills));

}

}

**Backspace String Compare**

import java.util.Stack;

public class Main {

public static int compareStrings(String s1, String s2) {

Stack<Character> t1 = new Stack<>();

Stack<Character> t2 = new Stack<>();

// Process the first string (s1)

for (char x : s1.toCharArray()) {

if (x == '#') {

if (!t1.isEmpty()) {

t1.pop();

}

} else {

t1.push(x);

}

}

// Process the second string (s2)

for (char x : s2.toCharArray()) {

if (x == '#') {

if (!t2.isEmpty()) {

t2.pop();

}

} else {

t2.push(x);

}

}

// Compare the two stacks

if (t1.equals(t2)) {

return 1;

}

return 0;

}

public static void main(String[] args) {

String s1 = "ab#c";

String s2 = "ad#c";

System.out.println(compareStrings(s1, s2)); // Output will be 1 if they are the same, 0 otherwise

}

}

**Beautiful String**

public class Main {

public static int getMinimumOperationCount(String s) {

int ans = 0;

int i = 1;

while (i < s.length()) {

if (Math.abs(s.charAt(i) - s.charAt(i - 1)) <= 1) {

ans++;

i++;

}

i++;

}

return ans;

}

public static void main(String[] args) {

String s = "abcde";

System.out.println(getMinimumOperationCount(s)); // Output the result

}

}

**Compression**

public class Main {

public static String compressedString(String message) {

int n = message.length();

int ind = 0;

StringBuilder ret = new StringBuilder();

while (ind < n) {

// Temporary index to count the number of consecutive occurrences of the character at index ind

int ind1 = ind;

while (ind1 < n && message.charAt(ind1) == message.charAt(ind)) {

ind1++;

}

// Add the current character to the return string

ret.append(message.charAt(ind));

// Append consecutive occurrence count if it's greater than 1

if (ind1 != ind + 1) {

ret.append(ind1 - ind);

}

// Move the index to the next new character

ind = ind1;

}

return ret.toString();

}

public static void main(String[] args) {

String message = "aaabbbcc";

System.out.println(compressedString(message)); // Output: "a3b3c2"

}

}

**Count Balanced Subarray**

public class Main {

public static int countBalancedSubarrays(int[] componentValue) {

int n = componentValue.length;

int even\_even\_cnt = 0;

int even\_odd\_cnt = 0;

int odd\_even\_cnt = 0;

int odd\_odd\_cnt = 0;

int ans = 0;

int cnt = 0;

for (int i = 0; i < n; i++) {

if (componentValue[i] % 2 == 1) {

cnt++;

}

if (i % 2 == 0) {

if (cnt % 2 == 1) {

ans += even\_even\_cnt + 1;

odd\_odd\_cnt++;

} else {

ans += even\_odd\_cnt;

odd\_even\_cnt++;

}

} else {

if (cnt % 2 == 1) {

ans += odd\_even\_cnt;

even\_odd\_cnt++;

} else {

ans += odd\_odd\_cnt;

even\_even\_cnt++;

}

}

}

return ans;

}

public static void main(String[] args) {

int[] componentValue = {1, 2, 3, 4};

System.out.println(countBalancedSubarrays(componentValue)); // Output the result

}

}

**Counting Triplets**

import java.util.HashMap;

import java.util.Map;

public class Main {

public static int getTripletCount(int[] a, int k) {

int n = a.length;

int ans = 0;

Map<Integer, Integer> freq = new HashMap<>();

// Calculate frequency of a[i] % k

for (int elem : a) {

int mod = elem % k;

freq.put(mod, freq.getOrDefault(mod, 0) + 1);

}

// Iterate through the array to find triplets

for (int i = 0; i < n; i++) {

// Decrease frequency for the current element (a[i] % k)

int modI = a[i] % k;

freq.put(modI, freq.get(modI) - 1);

for (int j = i + 1; j < n; j++) {

// Decrease frequency for the current element (a[j] % k)

int modJ = a[j] % k;

freq.put(modJ, freq.get(modJ) - 1);

// Check if the complement to form a triplet exists

int complement = (k - ((a[i] + a[j]) % k)) % k;

if (freq.containsKey(complement)) {

ans += freq.get(complement);

}

}

// Restore the frequency of a[j] % k after inner loop

for (int j = i + 1; j < n; j++) {

int modJ = a[j] % k;

freq.put(modJ, freq.get(modJ) + 1);

}

}

return ans;

}

public static void main(String[] args) {

int[] a = {1, 2, 3, 4};

int k = 5;

System.out.println(getTripletCount(a, k)); // Output the result

}

}

**Data Reorganization**

import java.util.Arrays;

public class Solution {

public int getMinimumValue(int[] data, int maxOperations) {

if (maxOperations >= 3) {

return 0;

}

Arrays.sort(data);

int n = data.length;

int d = data[0];

for (int i = 0; i < n - 1; i++) {

d = Math.min(d, data[i + 1] - data[i]);

}

if (maxOperations == 1) {

return d;

}

for (int i = 0; i < n; i++) {

for (int j = 0; j < i; j++) {

int v = data[i] - data[j];

int p = n;

for (int index = 0; index < n; index++) {

if (data[index] >= v) {

p = index;

break;

}

}

if (p < n) {

d = Math.min(d, data[p] - v);

}

if (p > 0) {

d = Math.min(d, v - data[p - 1]);

}

}

}

return d;

}

public static void main(String[] args) {

Solution solution = new Solution();

int[] data = {10, 20, 30};

int maxOperations = 2;

System.out.println(solution.getMinimumValue(data, maxOperations)); // Output will depend on the input data

}

}

**Data Transformation**

import java.util.ArrayList;

import java.util.List;

public class Solution {

public List<Integer> isConvertibleData(List<String> dataset) {

List<Integer> ans = new ArrayList<>();

for (String str : dataset) {

int mn = 0;

int psum = 0;

for (char ch : str.toCharArray()) {

if (ch == '(') {

psum += 1;

} else {

psum -= 1;

}

if (psum < mn) {

mn = psum;

}

}

if (psum != 0 || mn < -1) {

ans.add(0);

} else {

ans.add(1);

}

}

return ans;

}

public static void main(String[] args) {

Solution solution = new Solution();

List<String> dataset = new ArrayList<>();

dataset.add("(()())");

dataset.add("())(");

dataset.add("((())");

List<Integer> result = solution.isConvertibleData(dataset);

System.out.println(result); // Output: [1, 0, 0]

}

}

**Even Difference**

import java.util.Arrays;

public class Main {

public static int findLongestSubsequence(int[] arr) {

Arrays.sort(arr); // Sort the array

int firstOdd = -1, firstEven = -1, lastOdd = -1, lastEven = -1;

// Iterate over the array to find the first and last indices of odd and even numbers

for (int i = 0; i < arr.length; i++) {

if (arr[i] % 2 == 0) { // Even number

lastEven = i;

if (firstEven == -1) {

firstEven = i;

}

} else { // Odd number

lastOdd = i;

if (firstOdd == -1) {

firstOdd = i;

}

}

}

int ans = 1; // Default answer is 1

// Check the range for odd and even subsequences

if (firstOdd != -1) {

ans = Math.max(ans, lastOdd - firstOdd + 1);

}

if (firstEven != -1) {

ans = Math.max(ans, lastEven - firstEven + 1);

}

return ans;

}

public static void main(String[] args) {

int[] arr = {1, 3, 2, 4, 6, 7};

System.out.println(findLongestSubsequence(arr)); // Output the result

}

}

**Extraordinary Substrings**

public class Main {

public static int countSubstrings(String inputStr) {

int n = inputStr.length();

// Create an array to store the mapped values for all lowercase letters

int[] val = new int[26];

int x = 1;

for (int i = 0; i < 26; i++) {

// Mapped value remains the same for every 3 characters except for 1

if ((i + 1) % 3 == 0) {

x++;

}

val[i] += x;

}

// Count the number of extraordinary substrings

int count = 0;

for (int i = 0; i < n; i++) {

for (int j = i + 1; j <= n; j++) {

int sum = 0;

// Calculate the sum of mapped values

for (int k = i; k < j; k++) {

sum += val[inputStr.charAt(k) - 'a'];

}

// Check if the sum is divisible by the length of the substring

if (sum % (j - i) == 0) {

count++;

}

}

}

return count;

}

public static void main(String[] args) {

String inputStr = "abc";

System.out.println(countSubstrings(inputStr)); // Example usage

}

}

**Find the Sequence Sum**

public class Main {

public static int getSequenceSum(int i, int j, int k) {

int cntInc = j - i + 1; // Count of terms in the increasing sequence

int cntDec = j - k + 1; // Count of terms in the decreasing sequence

// Sum of the increasing sequence

int incSum = (cntInc \* (i + j)) / 2;

// Sum of the decreasing sequence

int decSum = (cntDec \* (j + k)) / 2;

// Total sum minus j to avoid double-counting

int totSum = incSum + decSum - j;

return totSum;

}

public static void main(String[] args) {

// Example usage of getSequenceSum

int i = 1, j = 5, k = 3;

System.out.println(getSequenceSum(i, j, k)); // Output the result

}

}

**Longest Even Length Word**

public class Main {

public static String longestEvenWord(String sentence) {

// Split the sentence into words

String[] words = sentence.split(" ");

String maxWord = "";

// Loop through all words

for (String word : words) {

// Check if the length of the word is even and greater than the current maxWord

if (word.length() % 2 == 0 && word.length() > maxWord.length()) {

maxWord = word;

}

}

// If no even length word was found, return "00"

if (maxWord.isEmpty()) {

maxWord = "00";

}

return maxWord;

}

public static void main(String[] args) {

// Example usage

String sentence = "I love programming at night";

System.out.println(longestEvenWord(sentence)); // Output the result

}

}

**Maximal Substring**

public class Main {

public static int getMaxSubstring(String dataStream) {

int c00 = 0;

int c01 = 0;

int c10 = 0;

int c11 = 0;

int n = dataStream.length();

// Ensure the length of dataStream is valid (even and within bounds)

if (n < 2 || n > 2 \* Math.pow(10, 5) || n % 2 != 0) {

throw new IllegalArgumentException("Invalid input size");

}

// Loop through the dataStream in pairs of characters

for (int i = 0; i < n; i += 2) {

if (dataStream.charAt(i) == '0' && dataStream.charAt(i + 1) == '0') {

c00++;

}

if (dataStream.charAt(i) == '0' && dataStream.charAt(i + 1) == '1') {

c01++;

}

if (dataStream.charAt(i) == '1' && dataStream.charAt(i + 1) == '0') {

c10++;

}

if (dataStream.charAt(i) == '1' && dataStream.charAt(i + 1) == '1') {

c11++;

}

}

// Calculate the result based on the counts

int ans = 2 \* c00 + 2 \* Math.min(1, c01) + Math.min(2, c10) + 2 \* c11;

return ans;

}

public static void main(String[] args) {

// Example usage

String dataStream = "0011001100";

System.out.println(getMaxSubstring(dataStream)); // Output the result

}

}

**Maximum Score**

import java.util.\*;

public class Main {

public static int getMaximumScore(int[] arr, int k) {

// Create a max-heap using a PriorityQueue with reverse order (to simulate max-heap)

PriorityQueue<Integer> maxHeap = new PriorityQueue<>(Collections.reverseOrder());

// Add all elements to the max-heap

for (int num : arr) {

maxHeap.offer(num);

}

int score = 0;

// Perform the operation k times

for (int i = 0; i < k; i++) {

// Get the largest element (highest absolute value)

int elem = maxHeap.poll();

// Add its absolute value to the score

score += Math.abs(elem);

// Modify the element as per the formula

elem = (Math.abs(elem) + 2) / 3;

// Push the modified element back into the max-heap (as a negative value to maintain max-heap)

maxHeap.offer(elem);

}

return score;

}

public static void main(String[] args) {

// Example usage

int[] arr = {1, 2, 3, 9, 8};

int k = 3;

System.out.println(getMaximumScore(arr, k)); // Output the result

}

}

**Memory Test**

import java.util.\*;

public class Main {

public static List<Integer> maxValues(int n, List<Integer> pos) {

// Find the maximum and minimum values in pos

int mx = Collections.max(pos);

int mn = Collections.min(pos);

// Create a result list to store the maximum differences

List<Integer> result = new ArrayList<>();

// For each index i from 0 to n-1, calculate the max of abs(mx - i) and abs(mn - i)

for (int i = 0; i < n; i++) {

int maxDiff = Math.max(Math.abs(mx - i), Math.abs(mn - i));

result.add(maxDiff);

}

return result;

}

public static void main(String[] args) {

// Example usage

List<Integer> pos = Arrays.asList(1, 3, 5, 7);

int n = 10;

System.out.println(maxValues(n, pos)); // Output the result

}

}

**Optimal Change**

public class Main {

public static int getMinInversion(String data, int maxFlips) {

int res = Integer.MAX\_VALUE;

int n = data.length();

// Helper function to compute the minimum inversion

Runnable func = new Runnable() {

@Override

public void run() {

int x = 0, y = 0, z = maxFlips;

// Count the number of '0's and their positions

for (int i = 0; i < n; i++) {

if (data.charAt(i) == '0') {

x++;

y += i;

}

}

// Traverse from the end and apply the max flips

for (int j = n - 1; j >= 0; j--) {

if (z == 0) {

break;

}

if (data.charAt(j) == '0') {

x--;

y -= j;

z--;

}

}

// Update the result with the calculated value

res = Math.min(res, y - (x \* (x - 1)) / 2);

}

};

func.run();

// Perform flips

StringBuilder sb = new StringBuilder(data);

for (int i = 0; i < n; i++) {

if (maxFlips == 0) {

break;

}

if (sb.charAt(i) == '1') {

sb.setCharAt(i, '0');

maxFlips--;

func.run();

}

}

return res;

}

public static void main(String[] args) {

String data = "010101";

int maxFlips = 2;

System.out.println(getMinInversion(data, maxFlips)); // Output the result

}

}

**Optimal Selection**

import java.util.\*;

public class Main {

public static int getMinimumIncrement(int[] arr) {

int n = arr.length;

// Check constraints (size and values within bounds)

if (n < 1 || n > 2 \* Math.pow(10, 5)) {

throw new IllegalArgumentException("Array size out of bounds");

}

for (int x : arr) {

if (x < 1 || x > Math.pow(10, 9)) {

throw new IllegalArgumentException("Array elements out of bounds");

}

}

// Append int(1e9 + 1) to arr

int[] newArr = Arrays.copyOf(arr, n + 1);

newArr[n] = (int) (1e9 + 1);

int l = 0;

int r = (int) (1e9 + 1);

// Iterate through the array to adjust the range [l, r]

for (int i = 1; i < n; i++) {

if (newArr[i] < newArr[i - 1]) {

l = Math.max(l, newArr[i - 1] - newArr[i]);

r = Math.min(r, newArr[i + 1] - newArr[i]);

}

}

// Return the result

if (l <= r) {

return l;

}

return -1;

}

public static void main(String[] args) {

// Example usage

int[] arr = {3, 5, 2, 8};

System.out.println(getMinimumIncrement(arr)); // Output the result

}

}

**Optimal Watchlist**

import java.util.\*;

public class Main {

public static int getMaxWatchlistScore(int[] duration, int[] rating, int limit) {

int n = duration.length;

// Create an array of tuples (rating, duration) and sort by rating in descending order

List<int[]> a = new ArrayList<>();

for (int i = 0; i < n; i++) {

a.add(new int[]{rating[i], duration[i]});

}

// Sort the list in descending order based on ratings

a.sort((x, y) -> Integer.compare(y[0], x[0]));

// Min-heap to store the durations

PriorityQueue<Integer> pq = new PriorityQueue<>();

int sum = 0;

int ans = 0;

// Iterate over the sorted list

for (int[] video : a) {

int min\_rat = video[0];

int time = video[1];

pq.offer(time);

sum += time;

// If the heap exceeds the limit, remove the smallest duration

if (pq.size() > limit) {

sum -= pq.poll();

}

// Update the maximum score

ans = Math.max(ans, sum \* min\_rat);

}

return ans;

}

public static void main(String[] args) {

// Example usage

int[] duration = {1, 2, 3, 4};

int[] rating = {5, 6, 7, 8};

int limit = 2;

System.out.println(getMaxWatchlistScore(duration, rating, limit)); // Output the result

}

}

**Pass the Baton**

public class Main {

public static int[] batonPass(int friends, int time) {

// Calculate how many full turns and remainder steps

int turns = time / (friends - 1);

int rem = time % (friends - 1);

int[] result = new int[2];

if (turns % 2 == 1) {

// If turns is odd, the baton passes in reverse order

if (rem == 0) {

result[0] = friends - 1;

result[1] = friends;

} else {

result[0] = friends - rem + 1;

result[1] = friends - rem;

}

} else {

// If turns is even, the baton passes in normal order

if (rem == 0) {

result[0] = 2;

result[1] = 1;

} else {

result[0] = rem;

result[1] = rem + 1;

}

}

return result;

}

public static void main(String[] args) {

// Example usage

int friends = 5;

int time = 8;

int[] result = batonPass(friends, time);

System.out.println("Baton passes between friend " + result[0] + " and friend " + result[1]);

}

}

**Shortest Substring**

import java.util.HashSet;

public class Main {

// Find the minimum length of a substring with repeated characters

public static int findMinLength(String s) {

int n = s.length();

boolean[] visited = new boolean[26];

int len = 0;

// Initialize visited array

for (int i = 0; i < 26; i++) {

visited[i] = false;

}

// Iterate through the string

for (int i = 0; i < n; i++) {

if (visited[s.charAt(i) - 'a']) {

for (int j = n - 1; j >= i; j--) {

if (visited[s.charAt(j) - 'a']) {

len = j - i + 1;

break;

} else {

visited[s.charAt(j) - 'a'] = true;

}

}

break;

} else {

visited[s.charAt(i) - 'a'] = true;

}

}

return len;

}

// Find the shortest substring with repeated characters

public static int findShortestSubstring(String s) {

int ans = findMinLength(s);

String reversed = new StringBuilder(s).reverse().toString();

ans = Math.min(ans, findMinLength(reversed));

return ans;

}

// Find the first repeating character in the string

public static int findFirstRepeating(String s, HashSet<Character> found) {

for (int i = 0; i < s.length(); i++) {

if (found.contains(s.charAt(i))) {

return i;

}

found.add(s.charAt(i));

}

return s.length();

}

// Another version of findShortestSubstring with the first repeating method

public static int findShortestSubstringUsingFirstRepeating(String s) {

int n = s.length();

HashSet<Character> found = new HashSet<>();

int l = findFirstRepeating(s, found);

found.clear();

int r = n - 1 - findFirstRepeating(new StringBuilder(s).reverse().toString(), new HashSet<>(found));

int ans = r - l + 1;

found.clear();

l = findFirstRepeating(new StringBuilder(s).reverse().toString(), found);

found.clear();

r = n - 1 - findFirstRepeating(s, new HashSet<>(found));

ans = Math.min(ans, r - l + 1);

return ans;

}

public static void main(String[] args) {

String s = "abca";

System.out.println("Shortest Substring Length (Method 1): " + findShortestSubstring(s)); // Example

System.out.println("Shortest Substring Length (Method 2): " + findShortestSubstringUsingFirstRepeating(s)); // Example

}

}

**Spam Detection**

import java.util.\*;

public class Main {

public static List<String> getSpamEmails(List<String> subjects, List<String> spamWords) {

// Convert spamWords to a HashSet for fast lookups

Set<String> spamWordsSet = new HashSet<>(spamWords);

List<String> ans = new ArrayList<>();

// Iterate over each subject

for (String subject : subjects) {

// Split the subject into words

String[] words = subject.split(" ");

int count = 0;

// Count how many words in the subject are in spamWordsSet

for (String word : words) {

if (spamWordsSet.contains(word)) {

count++;

}

}

// If the count is 2 or more, it's "spam", otherwise "not\_spam"

if (count >= 2) {

ans.add("spam");

} else {

ans.add("not\_spam");

}

}

return ans;

}

public static void main(String[] args) {

// Example usage

List<String> subjects = Arrays.asList("Free money now", "Meeting tomorrow", "Hurry, limited time offer", "Hello, how are you?");

List<String> spamWords = Arrays.asList("free", "offer", "money", "limited");

List<String> result = getSpamEmails(subjects, spamWords);

System.out.println(result); // Output the results

}

}

**Stock Prices**

import java.util.HashMap;

public class Main {

public static int getTripletCount(int[] arr, int d) {

int n = arr.length;

int res = 0;

// Frequency of modulo remainders

HashMap<Integer, Integer> moduloRemainders = new HashMap<>();

// Build the initial frequency map

for (int a : arr) {

int remainder = a % d;

moduloRemainders.put(remainder, moduloRemainders.getOrDefault(remainder, 0) + 1);

}

// Iterate over the array to check triplets

for (int i = 0; i < n; i++) {

int firstRemainder = arr[i] % d;

// Remove the current element from the frequency map

moduloRemainders.put(firstRemainder, moduloRemainders.get(firstRemainder) - 1);

// Check pairs of the form (arr[i], arr[j])

for (int j = i + 1; j < n; j++) {

int secondRemainder = arr[j] % d;

// Remove arr[j] from the frequency map

moduloRemainders.put(secondRemainder, moduloRemainders.get(secondRemainder) - 1);

// Calculate the necessary remainder to complete the triplet

int requiredRemainder = (d - (firstRemainder + secondRemainder) % d) % d;

if (moduloRemainders.containsKey(requiredRemainder)) {

res += moduloRemainders.get(requiredRemainder);

}

// Restore the frequency map for arr[j]

moduloRemainders.put(secondRemainder, moduloRemainders.get(secondRemainder) + 1);

}

// Restore the frequency map for arr[i]

moduloRemainders.put(firstRemainder, moduloRemainders.get(firstRemainder) + 1);

}

return res;

}

public static void main(String[] args) {

// Example usage

int[] arr = {1, 2, 3, 4, 5};

int d = 5;

System.out.println("Number of triplets: " + getTripletCount(arr, d)); // Example output

}

}

**Storing Processes**

public class Main {

public static int getMaximumStorageEfficiency(int[] numSegments, int m) {

int lo = 1, hi = Integer.MAX\_VALUE, ans = -1;

// Set the hi value to the minimum of numSegments

for (int x : numSegments) {

hi = Math.min(hi, x);

}

// Perform binary search

while (lo <= hi) {

int mid = (lo + hi) / 2;

int canFill = 0;

// Calculate how many smaller segments we can fill with size `mid`

for (int x : numSegments) {

canFill += x / mid;

}

// If we can fill at least m smaller segments, try for larger mid values

if (canFill >= m) {

ans = mid;

lo = mid + 1;

} else {

hi = mid - 1;

}

}

return ans;

}

public static void main(String[] args) {

// Example usage

int[] numSegments = {5, 10, 15, 20};

int m = 5;

System.out.println("Maximum Storage Efficiency: " + getMaximumStorageEfficiency(numSegments, m));

}

}

**Team Management**

import java.util.Arrays;

import java.util.Collections;

import java.util.List;

public class Main {

public static int getMaximumStrengthSum(int[] empSkill, List<Integer> teamSize) {

int n = empSkill.length;

// Calculate the total number of team members

int tot = 0;

for (int size : teamSize) {

tot += size;

}

// Sort the employee skills in ascending order

Arrays.sort(empSkill);

// Sort the team sizes in descending order

Collections.sort(teamSize, Collections.reverseOrder());

int ans = 0;

int st = 0, en = n - 1;

// Handle teams of size 1

while (!teamSize.isEmpty() && teamSize.get(teamSize.size() - 1) == 1) {

teamSize.remove(teamSize.size() - 1); // Pop the team size of 1

ans += 2 \* empSkill[en]; // Add double the skill of the highest skilled employee

en--;

}

// Reverse the team size list to handle larger teams

Collections.reverse(teamSize);

// Handle teams of size greater than 1

for (int size : teamSize) {

ans += empSkill[st] + empSkill[en]; // Add the lowest and highest skilled employees

en--;

st += size - 1; // Move the start pointer by the size of the team - 1

}

return ans;

}

public static void main(String[] args) {

// Example usage

int[] empSkill = {5, 1, 4, 2, 3};

List<Integer> teamSize = Arrays.asList(2, 2, 1);

System.out.println("Maximum Strength Sum: " + getMaximumStrengthSum(empSkill, teamSize)); // Example output

}

}

**Two Strings**

import java.util.HashSet;

import java.util.Set;

public class Solution {

public void commonSubstring(String[] a, String[] b) {

// Iterate through pairs of strings

for (int i = 0; i < a.length; i++) {

String s1 = a[i];

String s2 = b[i];

// Store characters of each string in sets

Set<Character> set1 = new HashSet<>();

Set<Character> set2 = new HashSet<>();

// Add each character from s1 to set1

for (char ch : s1.toCharArray()) {

set1.add(ch);

}

// Add each character from s2 to set2

for (char ch : s2.toCharArray()) {

set2.add(ch);

}

// Check if there is any intersection between the two sets

if (!set1.isEmpty() && !set2.isEmpty() && !set1.isEmpty() && set1.containsAny(set2) ){

System.out.println("YES");

} else {

System.out.println("NO");

}

}

}

public static void main(String[] args) {

Solution solution = new Solution();

String[] a = {"hello", "abc", "xyz"};

String[] b = {"world", "def", "pqr"};

solution.commonSubstring(a, b);

}

}

**Vowels**

import java.util.\*;

public class Main {

// Helper function to check if a character is a vowel

public static boolean checkVowel(char c) {

return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';

}

public static List<Integer> hasVowels(String[] strArr, List<String> query) {

int n = strArr.length;

// Prefix sum array to store count of strings that start and end with vowels

int[] pf = new int[n + 1];

// Build the prefix sum array

for (int i = 0; i < n; i++) {

if (checkVowel(strArr[i].charAt(0)) && checkVowel(strArr[i].charAt(strArr[i].length() - 1))) {

pf[i + 1] = 1;

}

pf[i + 1] += pf[i];

}

List<Integer> ans = new ArrayList<>();

// Process each query

for (String q : query) {

String[] curr\_q = q.split("-");

int l = Integer.parseInt(curr\_q[0]);

int r = Integer.parseInt(curr\_q[1]);

// Calculate the result for the current query using the prefix sum array

ans.add(pf[r] - pf[l - 1]);

}

return ans;

}

public static void main(String[] args) {

// Example usage

String[] strArr = {"apple", "banana", "orange", "umbrella", "kiwi"};

List<String> query = Arrays.asList("1-3", "2-5");

List<Integer> result = hasVowels(strArr, query);

System.out.println(result); // Expected output based on the example input

}

}

**Word Count Tool**

import java.util.\*;

public class Main {

public static int countValidWords(String s) {

int cnt = 0;

// Split the input string into words

String[] words = s.split(" ");

for (String word : words) {

word = word.toLowerCase(); // Convert to lowercase

// Check if the word meets the conditions

if (word.length() >= 3 && isAlphanumeric(word) && containsVowel(word) && containsConsonant(word)) {

cnt++;

}

}

return cnt;

}

// Helper function to check if a word is alphanumeric

public static boolean isAlphanumeric(String word) {

for (char c : word.toCharArray()) {

if (!Character.isLetterOrDigit(c)) {

return false;

}

}

return true;

}

// Helper function to check if a word contains at least one vowel

public static boolean containsVowel(String word) {

for (char c : word.toCharArray()) {

if ("aeiou".indexOf(c) != -1) { // Check if the character is a vowel

return true;

}

}

return false;

}

// Helper function to check if a word contains at least one consonant

public static boolean containsConsonant(String word) {

for (char c : word.toCharArray()) {

if (Character.isAlphabetic(c) && "aeiou".indexOf(c) == -1) { // If the character is not a vowel

return true;

}

}

return false;

}

public static void main(String[] args) {

// Example usage

String input = "Hello world! sky2 aei";

System.out.println(countValidWords(input)); // Example output

}

}

**Maximum Occurring Character**

Given a string, return the character that appears the maximum number of times in the string. The string will contain only *ASCII* characters, from the ranges ('a'-'z','A'-'Z','0'-'9'), and case matters. If there is a tie in the maximum number of times a character appears in the string, return the character that appears first in the string.

**Example**

*text = abbbaacc*

Both '*a'* and '*b'* occur *3* times in *text.*  Since '*a'* occurs earlier, *a* is the answer.

**Function Description**

Complete the function *maximumOccurringCharacter* in the editor below.

*maximumOccurringCharacter* has the following parameter:

*string text:*  the string to be operated upon

Returns

*char :* The most occurring character that appears first in the string.

**Constraints**

* 10 ≤ |*text*|≤ 104
* All characters are alphanumeric, in the ranges ('a'-'z','A'-'Z','0'-'9')

Input Format For Custom Testing

The first line contains a string, *text*, denoting the text to be analyzed.

Sample Case 0

**Sample Input For Custom Testing**

STDIN     Function

-----     --------

helloworld →  text = "helloworld"

**Sample Output**

l

**Explanation**

The character *'l'* occurs the most, *3* times in the string '*helloworld'*.

Sample Case 1

**Sample Input For Custom Testing**

STDIN     Function

-----     --------

abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz →  text = "abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz"

**Sample Output**

a

**Explanation**

All characters in the string '*abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz'*occur exactly twice. As *'a'* has the lowest index, the character *'a'* is the answer.

There is no answer for this question